

WEST Search History

DATE: Tuesday, March 16, 2004

Hide?	<u>Set</u> <u>Name</u>	<u>Query</u>	<u>Hit</u> <u>Count</u>
		<i>DB=USPT,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR</i>	
<input type="checkbox"/>	L25	l21 and bypass\$7	17
<input type="checkbox"/>	L24	l20 and bypass\$7	23
<input type="checkbox"/>	L23	l19 and bypass\$7	26
<input type="checkbox"/>	L22	l19 same bypass\$7	0
<input type="checkbox"/>	L21	l16 near3 (queue or fifo or buffer or memory or storage)	51
<input type="checkbox"/>	L20	l16 with (queue or fifo or buffer or memory or storage)	84
<input type="checkbox"/>	L19	l16 same (queue or fifo or buffer or memory or storage)	100
<input type="checkbox"/>	L18	L16.clm.	20
<input type="checkbox"/>	L17	L16.ab.	40
<input type="checkbox"/>	L16	(execut\$4 near3 (request or command or instruction) near3 without near3 stor\$4)	175
<input type="checkbox"/>	L15	L13 same (execut\$4 near3 (request or command or instruction) near3 without near3 stor\$4)	0
<input type="checkbox"/>	L14	L13 same (execut\$4 near3 (request or command or instruction))	120
<input type="checkbox"/>	L13	(bypass\$4 near5 (queue or buffer or fifo))	1996
<input type="checkbox"/>	L12	(bypass\$4 near5 (queue or buffer or fifo or stor\$4))	4798
<input type="checkbox"/>	L11	(hot-plug\$4 or hot-inser\$7) same rebuild\$4 same initializ\$8 same verif\$9	2
<input type="checkbox"/>	L10	L8 same (command or instruction or request\$4)	5
<input type="checkbox"/>	L9	L8 same request\$4	1
<input type="checkbox"/>	L8	(hot-plug\$4) same (queue or buffer)	12
<input type="checkbox"/>	L7	hot-plug\$4 same (requests near5 (queue or buffer))	0
<input type="checkbox"/>	L6	(hot-plug\$4) near5 (queue or buffer)	1
<input type="checkbox"/>	L5	(hot-plug adj event) near2 (queue or buffer)	0
<input type="checkbox"/>	L4	(irv near3 queues)	0
<input type="checkbox"/>	L3	(irv near2 queues)	0
<input type="checkbox"/>	L2	(IRV near2 queues)	0
<input type="checkbox"/>	L1	(queue near2 depth near2 check\$4)	18

END OF SEARCH HISTORY

First Hit Fwd Refs



Generate Collection

Print

L25: Entry 15 of 17

File: USPT

Jul 21, 1987

DOCUMENT-IDENTIFIER: US 4682284 A

**** See image for Certificate of Correction ****

TITLE: Queue administration method and apparatus

Abstract Text (1):

A memory subcontroller (203) of a computer (100) includes a queue (301) for storing read and write requests issued by memory using units (101-103) to a memory (104), apparatus (303, 304) for executing requests on the memory, and a circuit (302) for administering the queue. When the queue is empty and the executing apparatus is ready to receive a request for execution, a request incoming from a using unit bypasses the queue: it is received by the executing apparatus directly and is not stored in the queue. Otherwise, the queue administration circuit stores the request in the queue and then awaits results of validity checks on the stored request. If the request is found to be invalid, generally the administration circuit discards the request from the queue by freeing the queue location (210) or locations that store the invalid request to store the next received request. The invalid request is then overwritten by the next received request.

Brief Summary Text (6):

In conventional processing systems, task requestors and the task performer are allowed to communicate through the queue only. That is, a task requestor may not transfer a task to the task performer directly, by bypassing the queue, even when the task performer is idle and awaiting receipt of a task. The process of storing a task in the queue, then either propagating it through the FIFO queue or changing the pointers on a circular buffer queue, and finally retrieving a task from queue, takes time, and hence the system is slowed down by these procedures and its performance is adversely affected.

Brief Summary Text (11):

This invention is directed to solving these and other disadvantages of the prior art. According to the invention, a queuing arrangement that includes a storage apparatus organized as a queue and including a plurality of storage locations for holding queue entries and apparatus for storing a queue entry in at least one location at the tail end of the queue, further includes an arrangement for determining the validity of the entry stored at the tail end and apparatus for freeing the at least one location if it is determined to hold an invalid tail end entry, to hold another queue entry. Stored queue entries are retrieved, for example for processing from the head end of the queue when the queue is not empty, and preferably the queue is bypassed and entries for processing are obtained directly when the queue is empty.

Brief Summary Text (16):

According to an aspect of the invention, the system described previously further includes an apparatus for determining whether the queue is empty, and apparatus for executing tasks on the resource, which apparatus is coupled to the retrieving apparatus for receiving therefrom retrieved tasks for execution and which apparatus is also arranged to receive tasks for execution from the requesting units directly, i.e., not through the queue. The executing apparatus causes the retrieving apparatus to retrieve a stored request when the determining apparatus finds the queue not empty, but receives a resource request directly from a using unit and

causes the storing apparatus not to store the received request when the determining apparatus finds the queue empty. Hence a task is enabled to bypass the queue and reach the task executor directly from the task requestor, thereby advantageously avoiding the loss of time involved in the idle task executor waiting for the task to be first stored in and then retrieved from the queue.

Detailed Description Text (68):

If the queue 301 is empty, i.e., the QE line 421 is asserted, the circuit 400 asserts the SCRTI signal line 428 leading to the queue administration circuit 302 to inform it that the circuit 303 is ready to take input directly from the subcontroller bus 208, and causes each of the multiplexers 320, 322, 401, and 403 to connect the respective bus 410-413 of the subcontroller bus 208 to its output port. A memory request, or a portion thereof, is permitted thereby to bypass the queue 301 and to reach the address and data handler 304 and the memory administration circuit 303 directly from the subcontroller bus 208.

CLAIMS:

1. An arrangement comprising:

storage means organized as a queue having a head end and a tail end, the storage means including a plurality of storage locations for holding queue entries;

means for storing a queue entry received for processing, in a free at least one location at the tail end of the queue;

means for determining whether the stored tail end entry is invalid;

means responsive to the determining means for freeing the at least one location holding an invalid tail end entry to hold a new tail entry;

means for retrieving a stored queue entry for processing from the head end of the queue;

means for determining whether the queue is empty; and

means, coupled to the retrieving means and cooperative with the means for determining whether the queue is empty, for obtaining a retrieved entry from the retrieving means when the queue is determined not to be empty, and for receiving an entry for processing directly, by bypassing the queue, when the queue is determined to be empty.

33. The method of claim 29 in a system further having apparatus for indicating when the queue is empty and apparatus for executing requests on the resource and for indicating readiness to receive a request for execution, wherein

the steps of storing, determining, and causing the tail end pointer to point, are performed in response to indication that the executing apparatus is not ready to receive a request;

wherein the steps of retrieving, causing the head end pointer to point, and selectively repeating, are performed in response to indication that the queue is not empty and the executing apparatus is ready to receive a request; and

the method further comprising the step of

executing a resource request received from a unit, without storing the request in the queue, in response to indication that the queue is empty and the executing apparatus is ready to receive a request.

First Hit Fwd Refs

Generate Collection

Print

L1: Entry 2 of 18

File: USPT

Jan 13, 2004

DOCUMENT-IDENTIFIER: US 6678271 B1

TITLE: High performance system and method having a local bus and a global bus

Detailed Description Text (43):

The determination to drop or pass a traffic unit results from the execution of a threshold method. There is a threshold method performed for both queues. During the ingress threshold method: 1) the local bus gateway 402 accepts traffic units from the local bus 406 destined for the global bus 407; 2) the local bus gateway 402 signifies to block 411 (via local bus gateway control lines 410) that an ingress traffic unit has arrived; 3) the traffic control logic associated with block 411 compares the ingress state register 414 value (i.e., how many traffic units are currently in the ingress queue) against its knowledge of the depth the ingress queue (e.g., by checking the queue depth register 415 or determining the ingress queue depth from the value stored in the queue depth register 415) to make a determination whether or not there is room in the ingress queue for the newly arrived traffic unit.

Detailed Description Text (44):

The egress threshold method operates in a similar manner. That is: 1) the global bus gateway 403 accepts traffic units from the global bus 407 destined for the local bus 406; 2) the global bus gateway 403 signifies to block 411 (via global bus gateway control lines 418) that a traffic unit has arrived; 3) the traffic control logic associated with block 411 compares the egress state register 416 value (i.e., how many traffic units are currently in the egress queue) against its knowledge of the depth the egress queue (e.g., by checking the queue depth register 415 or determining the egress queue depth from the value stored in the queue depth register 415) to make a determination whether or not there is room in the egress queue for the newly arrived traffic unit.

[First Hit](#) [Fwd Refs](#)

Generate Collection

Print

L1: Entry 3 of 18

File: USPT

Dec 23, 2003

DOCUMENT-IDENTIFIER: US 6667975 B1

TITLE: Port aggregation load balancing

Detailed Description Text (26):

Whenever an element is pushed onto a transmit queue, the least utilized queue process for each port group compares the queue number with the least utilized queue register. If the numbers are different, it ignores the push event. If, however, the numbers are the same, it checks transmit queue depth register for that queue. If the transmit queue depth register is zero, the process does nothing. If the transmit queue depth register is non-zero, it checks to see if any other queues in the port group have no entries. If so, then it selects the next queue, from those with no entries, to be the new "least utilized queue." If all of the other queues have entries, the process does nothing.

[First Hit](#) [Fwd Refs](#)☐ [Generate Collection](#) [Print](#)

L1: Entry 4 of 18

File: USPT

Aug 12, 2003

DOCUMENT-IDENTIFIER: US 6606630 B1

TITLE: Data structure and method for tracking network topology in a fiber channel port driver

Detailed Description Text (57):

When I/O operation exchanges terminate 1000 (FIG. 10), frames returned by the target node are tested 1002 for queue overflow errors. If a queue overflow error occurs, its time is compared to the last queue decrement time 628, the prior contents of the last queue decrement time being used to permit downward adjustment of queue depth at a maximum rate of once every five seconds. If downward adjustment of queue depth is done, the last queue decrement time 628 is updated. The current queue depth 622 is checked 1004 to determine if it is already at the minimum value. If a queue overflow error occurs with the queue depth at other than the minimum value, the current queue depth 622 is adjusted downwardly. The downward adjustment is either by five 1010, or to the minimum value 1006 or 1008, depending on the value of the queue depth and the prior contents of the last queue refusal variable.

[First Hit](#) [Fwd Refs](#)**End of Result Set**

Generate Collection

Print

L9: Entry 1 of 1

File: USPT

Nov 26, 2002

DOCUMENT-IDENTIFIER: US 6487623 B1

TITLE: Replacement, upgrade and/or addition of hot-pluggable components in a computer system

Detailed Description Text (28):

Data from the present RAM 106 module must be synchronized with the new RAM 106 module. The data stored in the present RAM 106 module must be transferred in a background mode to the new RAM 106 module as described hereinabove for replacing a failing component. Once all of the data from the present RAM 106 module has been written to the new RAM 106 module, the hot-plug controller 164 will synchronously disconnect the appropriate connector 402 from the memory bus 105a with the FET signal isolation buffers 160, and then disconnect system power from this connector 402 with the power FET switches 162. The hot-plug controller 164 will also enable the new RAM 106 module to respond to read request. The memory capacity which is in excess of the memory capacity of the RAM 106 module being replaced, needs to be initialized by writing all zeros to any memory locations not mapped from the RAM 106 module being replaced. After the synchronization and initialization process has been completed, the technician can remove the disabled RAM 106 module(s), thus freeing up more connectors 402 for adding more new RAM 106 modules in the future.

First Hit Fwd Refs

Generate Collection

Print

L10: Entry 4 of 5

File: USPT

Aug 28, 2001

DOCUMENT-IDENTIFIER: US 6282596 B1

TITLE: Method and system for hot-plugging a processor into a data processing system

Detailed Description Text (24):

Block 112 depicts setting the power-on configuration features for the processor card of the particular processor subsystem. These functions are accomplished by the service processor which sends commands to the hot-plug controller through I2C. The hot-plug controller then directly drives a group of processor signal pins through CFG to the processor card and samples the results of the data passing therethrough. Features such as advanced programmable interrupt controller (APIC) identification (ID) selection are executed, where the APIC ID selection defines the APIC cluster group and symmetric multiprocessing (SMP) agent ID which are utilized for front-side bus arbitration. In addition, the processor core clock frequency ratio, processor in-order queue (IOQ) depth, processor observance of various error signals, processor power-on option to run BIST and others are among configurable options needed to be handled by the service processor.

First Hit Fwd Refs



Generate Collection

Print

L11: Entry 1 of 2

File: USPT

Jan 27, 2004

DOCUMENT-IDENTIFIER: US 6684292 B2

TITLE: Memory module resync

Detailed Description Text (53):

The hot-add procedure may be implemented if there is an empty slot on the memory cartridge 25A-25E and if software support exists. To implement the hot-add procedure, the user performs a normal hot-plug procedure on each of the memory cartridges 25A-25E and adds the same size DIMM to the same empty slot across all memory cartridges 25A-25E. If more than one DIMM per cartridge 25A-25E is added, the DIMMs added to the same memory cartridge 25A-25E do not need to be the same. However, if the current memory is striped, and memory striping is to be maintained on the new memory, the new memory is typically added in identical pairs across the two memory controller channels. Adhering to the striping rules will facilitate a hot-upgrade in the future. Next, the system rebuilds and verifies the current memory as each of the memory cartridges 25A-25E is hot-plugged into the system. Additional memory capacity may not be available until each of the memory cartridges 25A-25E is upgraded. Once all of the new memory (DIMMs) is added, it is initialized to a known value. This procedure may be initiated automatically or through a user interface. Once the additional memory is initialized, the operating system is informed that the additional memory is available.

Detailed Description Text (56):

To implement the hot-upgrade procedure, the user first verifies that the system is in a state that allows hot-upgrade. The user then determines the current memory configuration by part-number and DIMM-slot. Next, the user implements a normal hot-plug procedure, as described above, on the first memory cartridge to replace the smaller DIMM with the larger DIMM. The system will rebuild and verify the contiguous portion of the new DIMM. Next, the memory is brought online and the entire memory subsystem begins to operate in the redundant state using the new DIMM, until the system again enters the non-redundant state to upgrade the next memory cartridge. It should be understood that the additional memory capacity may not be available until all memory cartridges 25A-25E have been upgraded. Once the final memory cartridge 25A-25E is upgraded, the additional memory is initialized and the operating system is informed that additional memory is available so that the memory subsystem may return to normal operation.

Detailed Description Paragraph Table (6):

TABLE 6 Hot-plug Power-up Sequence HW SW .DELTA.t Description 1 Ext. Logic The M3PAL detects the IRS-long pin connection on the connector. The PAL will assert the PWRON_signal to the power controller. When the power controller sees the PWRON_signal asserted, it will turn the external FETs on to provide power to the cartridge. 1 Ext. Logic The power controller senses the voltage level on the output side of the FETs. When the voltage reaches .about.2.95 V, the power controller will deassert the PWRFAULT_signal. The M3PAL detects the PWRFAULT_signal deassertion and asserts the CLKEN signal. The CLKEN signal will enable the quick switch to connect the system clock to the cartridge. 1 HC t.sub.SCAN Detect a transition on the IRS-short signal indicating that a memory cartridge has been installed. 2 HC Generates an interrupt indicating a new memory cartridge has been installed. 3 SW Write to clear HC IRS status (HC, f0, A4-AC, bit1). 3 SW Write to HC to "blink once" all memory cartridge LED's for power-on test (HC, f0, 94-9C). 4 HC t.sub.SCANIN Detect

a transition on the PIRN signal indicating that the memory cartridge is ready for power-up. 5 HC Generates an interrupt indicating a new memory cartridge is ready to be powered up. 5 SW Write to clear HC PIRN status (HC, f0, A4-AC, bit0). 6 SW Writes to HC to turn Power LED to the blink state on the added memory cartridge (HC, f0, 94-9C, bit6-5). 7 SW Writes to the HC Auto Power On configuration register (HC, f0, D2, bit4-0). 8 HC t.sub.PU + t.sub.SCANOUT Asserts a signal indicating to an external device to enable power to the memory cartridge. 9 HC Load counter with Power Up Delay register value and begin count (HC, f0, E6). 10 HC Wait for count to expire. 11 HC t.sub.PLL + t.sub.SCANOUT Asserts a signal indicating to an external device to enable the system clock to memory cartridge and wait for PLL to lock. 12 HC Load counter with Clock Enable Delay register value and begin count (HC, f0, E8). 13 HC Wait for count to expire. 14 HC Signal to DC to execute power-up sequence. 15 DC Tristate MNET inputs. 16 HC t.sub.HC Tristate MNET inputs (if driving 0's when bus is disabled). 17 HC Drive RESET inactive to hot-plugged memory cartridge. 18 HC t.sub.SYNC Synchronize the MNET bus interface between HC, DC, and MC for the hot-plugged cartridge. 19 HC Generates an interrupt indicating that the power-on sequence is complete (HC, f0, D3, bit4-0). 20 SW Clear the Auto Power Up status register (HC, f0, D3, bit4-0). 20 SW Program I2C configuration registers (MC, f0, E8). 20 SW Program MC I2C configuration registers to execute reads of DIMM SPD register space (MC, f0, E0, E4, F0, F4). 21 SW t.sub.SPD MC executes I2C reads of up to 64-bits per programmed read and generates an interrupt (MC, f0, D8, bit4-0). 21 MC Write to clear IIC status (MC, f0, D8, bit4-0). 22 SW Verify DIMM compatibility of the hot-plugged memory cartridge with the other memory cartridges. If DIMM compatibility fails, see DIMM Imcompatibility section. 23 SW If initiating a hot-upgrade sequence, program all MC's to be in non-pipelined mode (MC, f0, C8, bit1). 24 SW Program MC configuration space on the hot-plugged memory cartridge. 25 SW Update CAS latency in other MC's if necessary. If CAS latency is changed, must set the Hot-reset MRS configuration bit so that an MRS will be executed on the MNET RESYNC (MC, f0, C8, bit12). 26 SW Update memory controller timing attributes in other MC's if necessary. Changes to memory controller attributes will not be seen by internal logic until HC generates an MNET RESYNC command (MC, f0, CC, D0). 27 SW Write to MC Initialize memory configuration register (MC, f0, C8, bit9). 28 MC t.sub.INITMEM Memory controller executes a precharge. 29 MC Memory controller executes a Mode Register Set cycle. 30 MC Memory controller executes 8 refresh transactions. 31 SW Set the HC Flush and Resync configuration register (HC, f0, 90, bit4). 32 HC t.sub.FLUSH Flushes all the outstanding memory cycles to the MC's. 33 HC Generates an MNET RESYNC command to resynchronize all of the MC's. 34 MC-all Updates Memory Controller Attribute registers with shadow register program values, resets queue pointers and state machines, disables and re-enables the refresh counters to sync refresh, enables memory transactions, executes MRS if enabled. 35 SW Clear the HC Flush and Resync configuration register (HC, f0, 90, bit4). 36 SW Execute the Rebuild procedure (HC, f0, 90, bit0). 36 HC t.sub.REBUILD Execute Rebuild procedure and generate an interrupt (HC, f0, 92, bit0). 37 SW Enable data ECC and XOR compare logic in DC, auto-replace still enabled. 38 SW Place DC in Verify mode (DC, f0, 62, bit1). 39 SW Execute the Verify procedure (HC, f0, 90, bit2). See Verify Procedure section for details on handling errors during the Verify procedure. 39 HC t.sub.VERIFY Execute Verify procedure and generate an interrupt (HC, f0, 92, bit0). 40 SW Turn off XOR engine auto-replace, system fully redundant (DC, f0, 62, bit 1). 41 SW Write to HC to turn Power LED to "on" state on the added memory cartridge (HC, f0, 94-9C, bit6-5). 42 SW Write to HC to set the Fault LED to the "off" state for the powered up memory cartridge if previously indicating a fault condition (HC, f0, 94-9C, bit 8-7). 43 If a full set of additional memory added (hot-add, hot-upgrade): Execute Initialize procedure across new memory. Execute Verify procedure across new memory. 44 .DELTA.t Timing parameters: Parameter Value Description t.sub.SCANIN 200 usec. time required to bring input in through the scan chain, this does not account for debounce time t.sub.SCANOUT 200 usec. time required to drive an output through the scan chain t.sub.PU time required to enable power to the cartridge t.sub.PLL 10 usec. time required to lock PLL t.sub.DC 100 nsec. time required for DC to complete sequence t.sub.HC 100 nsec. time required for HC to complete sequence t.sub.SYNC 2 usec.